Aalto University
School of Business

USING NEURAL NETWORKS TO IMPROVE CROSS-SECTIONAL MOMENTUM STRATEGY PERFORMANCE

Master's Thesis

Leevi Lindgren
Aalto University School of Business
Finance

Spring 2019

Author	Leevi Lindgren
Title of thesis	Using neural networks to improve cross-sectional momentum strategy performance
Degree	Master of Science in Economics and Business Administration
Degree programme	Finance
Thesis advisor	Peter Nyberg
Year of approval	2019
Pages	39 + 5
Languange	English

Abstract

Machine learning and in particular, neural networks, are powerful tools for predictive analysis and used in a wide range of applications. I combine the neural networks and momentum investing with the aim to construct a neural network enhanced momentum strategy. I train the neural network to predict future relative performance of stocks in US between 1964 and 2012 using past returns and risk indicators as a feature set. I find that neural network momentum strategy generates monthly average returns of 3.18% (t-stat 11.81), outperforming the traditional momentum strategy by 1.39%-points. I also find that long-term momentum is the main driver of the relative performance predictions and that short and mid-term historical volatilities are the most important risk factors explaining variation in the predictions.

Keywords	neural networks, momentum, momentum crashes

Tekijä	Leevi Lindgren
Työn nimi	Neuroverkkojen hyödyntäminen momentum-sijoittamisessa
Tutkinto	Kauppatieteiden maisteri
Koulutusohjelma	Rahoitus
Työn ohjaaja	Peter Nyberg
Hyväksymisvuosi	2019
Sivumäärä	39 + 5
Kieli	Englanti

Abstrakti

Koneoppiminen ja erityisesti neuroverkot ovat tehokkaita metodeja ennustamiseen tarkoitettuihin tehtäviin ja niitä käytetään laajasti erityyppisissä sovelluksissa. Tässä tutkimuksessa yhdistän neuroverkot momentum-sijoittamisen kannsa, tarkoituksenani rakentaa neuroverkoilla parannettu momentum-strategia. Opetan neuroverkon ennustamaan Yhdysvaltalaisten osakkeiden relatiivista tuottokehitystä aikavälillä 1964-2012. Mallin muuttujina käytän osakkeiden historiallisia tuottoja, sekä useita erityyppisiä osakekohtaisia riski-indikaattoreita. Neuroverkoilla parannellun momentum-strategian keskimääräinen kuukausituotto on 3.18% (t-arvo 11.81), joka on 1.39 prosenttiyksikköä enemmän kuin perinteisellä momentum-strategialla. Lisäksi näytän, että pitkän aikavälin momentum on tärkein selittävä tekijä neuroverkon tekemien ennustusten variaatiossa. Vastaavasti, keskipitkän aikavälin historiallinen volatiliteetti on tärkein riski-indikaattori selittämään neuroverkon tekemien ennustusten variaatiota.

Avainsanat	neuroverkot, momentum, momentumromahdukset

Contents

1 Introduction

The recent developments in software and hardware technology in addition with the improved access to large, high quality datasets allows the use of more complex and advanced models. In particular, machine learning techniques are nowdays used in a vast range of industries and research areas. The goal of this paper is to bring together these exciting techniques with empirical finance and in particular, momentum investing.

In the context of finance, momentum is a phenomenon where assets that have performed well in the past tend to outperform those that have performed poorly. The goal of momentum investing is to exploit this anomaly. Momentum strategy is implemented so that relative performance of assets in some recent time window, e.g. 12 months, is measured and based on it assets are allocated into winner and loser portfolios. Then, a zero-investment long-short portfolio is constructed. Despite of its simplicity, momentum strategy has been profitable in multiple asset classes and time periods. For instance, Asness, Moskowitz, and Pedersen (2013) find momentum premium from eight different markets and asset classes and Goetzmann and Huang (2018) show that it can be found even from 19th century data from imperial Russia. For the rest of this paper, whenever I refer to momentum, I mean the cross-sectional momentum, not time-series momentum.[1]

Even though momentum strategies have generated high abnormal returns in the past, the strategy suffers from occasional, large negative returns. This is documented by Daniel and Moskowitz (2016), who show that these large negative returns, momentum crashes, often occur contemporaneously with market rebounds after market has declined and when ex-ante measures of volatility are high. These crashes are mostly driven by the short leg of the strategy. However, their paper among others such as Barroso and Santa-Clara (2015), show that these crashes are partly predictable. This motivates a class of risk adjusted momentum strategies, which try to improve the performance by using volatility as a risk measure and then scaling the strategy's exposure based on it.

The aim of my study is to use machine learning techniques to enhance cross-sectional momentum strategy performance. The particular machine learning methods I consider are artificial neural networks, which arguably are among the most powerful tools for predictive modeling. Currently, they are used in face recognition, image processing and medical diagnosis to name few applications. The strength of neural networks is their ability to approximate complex nonlinear associations and interactions in data.

Many of the papers combining machine learning and empirical finance use very large sets of features[2] to predict future returns and then sort stocks into portfolios such that

[1] For research related to time-series momentum, or equivalently, trend following, see e.g. Moskowitz, Ooi, and Pedersen (2012)

[2] The term "feature" is used interchangeably with an input variable of a model

stocks with high predicted returns are in the first portfolio and stocks with the lowest predicted returns in the last. Then, a zero-investment long-short portfolio is formed. As my intention is to study the cross-sectional momentum strategy, I have a somewhat different approach: I try to predict whether an individual stock is a winner or a loser instead of predicting directly future returns. I emphasize that my focus is on the relative performance, not on absolute return level. Also, the feature set I use consists only of past returns from different time spans, which are traditionally used to construct momentum portfolios, and stock level risk indicators. The use of risk indicators is motivated by the literature that suggests that momentum strategy's performance can be improved significantly by relatively simple risk management methods.

Neural network momentum strategy is constructed in a similar way as the traditional cross-sectional momentum strategy. To form the neural network enhanced momentum portfolio, I allocate stocks into winner and loser portfolios based on predicted relative performance of of each stock. Then, as in the benchmark momentum strategy, I construct zero-investment long-short portfolio. This methodology is similar to the one used by Takeuchi and Lee (2013) who analyze cross-sectional momentum strategies using restricted Boltzmann machine, a special type of neural network. However, they do not include any risk indicators in their feature set. I also use a wider time window for the strategy evaluation.

I also try to provide insight to which of the features drive the cross-sectional momentum and how the importances of different features vary over time. Unfortunately, neural networks are perhaps one of less transparent machine learning models and there is no similar interpretation of the feature importances that for instance OLS-regression coefficients provide. Hence, I use an alternative approach by measuring the importance of each feature by separately changing the value of each feature to zero and then comparing how much the predictions deviate from the predictions that are made with the full set of features.

I find that neural network enhanced cross-sectional momentum strategy outperforms benchmark momentum strategy by a wide margin: it increases average monthly return from 1.79% to 3.18% and annualized Sharpe ratio from 0.73 to 1.69. The results also suggest that neural network is also able to reduce the amount of negative extreme returns. Moreover, neural network momentum strategy generates a significant positive alpha after controlling for various risk factors. Also, I find that not surprisingly, historical momentum signals and volatility estimates are mostly driving the predictions. However, I am not able to distinguish time series variation in the feature importances.

Rest of this paper is organized as follows. I first give an overview of the related literature. Next, I provide a brief introduction to neural networks and describe the data I use and how

the set of features is constructed. Moreover, I demonstrate the process of how predictions are made and momentum portfolios constructed. Then, I report the results and analyze them and study which of the features are driving the predictions. Finally, I provide concluding remarks. Appendix contains details on the used optimization algorithms and provides additional results.

2 Literature review

2.1 Momentum

Momentum, the tendency of past winners to continue to outperform also in the future, is one of the most documented and studied anomalies in financial markets. It has been pervasive in various time periods and asset classes. Jegadeesh and Titman (1993) were the first to document the profitability of momentum strategies in U.S. stock market during 1965-1989. After that, momentum has received an extensive attention, but despite of that, there is still an active debate about its main drivers. Various explanations for momentum premium has been proposed in the academia, which can be divided into a risk and behavioral based evidence.

As examples of the latter, Grinblatt and Han (2005) show that some investors' tendency to hold on losing stocks and realize profits too soon is driven by mental accounting and prospect theory. Consequently, losing stocks become overvalued and winners overvalued. In other words, this means that past losers (winners) have low (high) expected returns. Moreover, George and Hwang (2004) show that momentum returns are driven by 52-week high price. Intuition behind their result is that after good news occur and stock price reaches near 52-week high price, investors are reluctant to bid the price higher even tough the information warrants otherwise. Eventually information prevails and price moves towards its fundamental value causing price momentum. A similar logic applies when negative news arrive. Daniel, Hirshleifer, and Subrahmanyam (1998) propose a theory of securities market under- and overreactions and suggest that biased self-attribution creates a positive short lag autocorrelation (i.e. momentum) in returns. Their idea is the following. Assuming that an investor trades based on a private signal, later the public signal confirms the trade if it has the same sign as the private signal. They assume that when the investor receives confirmation, his confidence increases but disconfirming public information causes confidence to fall only modestly. Hence, new public signals are viewed on average as confirming the validity of his private signals, meaning that new public information can trigger overreaction in security prices consequently causing price momentum.

Johnson (2002) provides evidence for the risk-based explanation. He proposes a simple, single firm model and shows that expected returns are a function of firm growth rates. Hence, a positive return shock provides a signal that the future growth prospects of firm's cash flows have improved causing the long-term expected returns to increase. Asness (1997) studies the relationship between momentum and value strategy and shows that momentum is particularly strong within growth stocks with risky future cash flows. Moreover, Daniel and Moskowitz (2016) as well as Ruenzi and Weigert (2018) show that compensation for tail risk in momentum strategies might be one explanation for the anomaly. The former investigate so-called momentum crashes and the latter build a marketwide crash risk factor which explains a large portion of the abnormal returns of momentum strategy.

2.2 Momentum crashes

As noted earlier, momentum strategies suffer from occasional crashes. Daniel and Moskowitz (2016) provide a comprehensive analysis of the matter and conclude that the crashes the momentum portfolio exhibits are associated with a preceding bear market period and a simultaneous market rebound. An intuitive explanation for this is that if market has fallen significantly during the momentum formation period, the strategy will be long on low-beta stocks and short on high-beta stocks. When the market rebounds, so will the high beta stocks and the short leg will generate extreme losses. In particular, they find that following major market declines, market betas for the losers stocks may raise above 3 and below 0.5 for winners. Their evidence suggests that the short leg of the strategy is the main driver of momentum crashes.

Observation of momentum crashes has motivated a line of research where methods to manage the crash risk of momentum strategy have been suggested. To manage the crash risk, various papers have proposed methods where volatility is used as the proxy of risk. Barroso and Santa-Clara (2015) introduce a constant volatility momentum strategy, which improves momentum performance. Daniel and Moskowitz (2016) use dynamic weighting where the strategy weight is adjusted based on ex-ante Sharpe ratio estimates. Between January 1934 and December 2012 benchmark momentum, constant volatility and dynamic weighting strategies yield annualized Sharpe ratios of 0.682, 1.041 and 1.194 respectively. These findings suggest that risk management methods improve the strategy performance and to some extent help to avoid the occasional crashes.

Recent papers, e.g. Kelly and Jiang (2014) and Chabi-Yo, Ruenzi, and Weigert (2018), have shown that investors dislike lower tail sensitive assets. Motivated by these results Ruenzi and Weigert (2018) investigate whether momentum returns are driven by strategy's exposure to crash risk. They use lower tail dependence (LTD) factor by Chabi-Yo et al.

(2018), which is a zero-investment strategy going long (short) in stocks with high (low) crash sensitivity, to explain momentum returns. They find that between years 1963 and 2012 in US, momentum's annual alpha is reduced from significant 11.94% to insignificant 1.84% after controlling for LTD factor. Their result is in line with the idea that momentum strategy's high risk adjusted returns have been compensation for crash sensitivity and provides further evidence for the risk-based explanation of the momentum anomaly.

2.3 Machine learning and finance

The term machine learning is often somewhat inexact and context specific, but I will assume the specification by Gu, Kelly, and Xiu (2018) and use the term to describe a diverse collection of high-dimensional models for prediction, combined with regularization methods to avoid overfitting the model and algorithms to find the best model from a vast number of different model specifications.

Machine learning methods have not yet received an extensive attention in financial literature. There are however two distinguishable lines of research. The first focuses on variable selection and dimension reduction techniques. Given that there are hundreds of different proposed pricing factors that are claimed to be driving the cross-sectional differences in expected returns, some researchers have shifted their focus on how to find the important variables from the vast set of different factors. Machine learning techniques such as least absolute shrinkage and selection operator (LASSO) regression and principal component methods are well suited for such task. For instance, Feng, Giglio, and Xiu (2017) propose a model selection method based on LASSO regression to evaluate contribution of a new factor to asset pricing. Similarly Messmer and Audrino (2017) use an adaptive LASSO for assessing factor importance in cross-section of expected returns. Moreover, Gu, Kelly, and Xiu (2019) propose an autoencoder asset pricing model which shrinkages wide range of firm level characteristics into a small number of factors. It can be interpreted as a nonlinear extension to the principal component decomposition.

In the second line of research, various papers lever the predictive power of machine learning methods to estimate the quantity $E\left(r_{i,t+1}|\mathcal{F}_t\right)$ where r_{t+1} and $\mathcal{F}_t$ are return of asset i in period $t+1$ excess of risk-free rate and information available to investor at time t respectively.[3] Even though literature suggests that it is undeniable that machine learning methods has great potential to improve these predictions, Gu et al. (2018) note that machine learning do not necessary tell us anything about equilibrium or economic mechanism driving the excess returns, which is probably one of the reasons for the limited number of papers studying machine learning and asset pricing.

[3]See e.g. Gu et al. (2018), Messmer (2017),

Perhaps the most comprehensive look at the use of machine learning in empirical finance is provided by Gu et al. (2018). Using US equity data with a large number of firm level characteristics and macroeconomic variables, they test the predictive power of a number of different machine learning methods. Their analysis suggests that all the tested machine learning models are able to beat the three factor model by Fama and French (1993), when performance is measured by out-of-sample prediction power. Also one of their findings is that methods that are able to capture the nonlinearities and interactions that financial data usually exhibits, perform the best. Such methods include for instance boosted trees and neural networks, which also motivates the use of neural networks in my study. For instance, Cybenko (1989) and Hornik (1991) show that neural networks can approximate any function. This means that if neural networks are used, any limitations to the form on which the function maps inputs as a prediction is not set, as the neural network can assume any functional form. This is not the case with linear models such as Fama-French three factor model (Fama & French, 1993).

3 Neural Networks

This section provides a brief introduction of neural networks and how their parameters and hyperparameters are estimated. Moreover, I introduce various regularization methods to avoid overfitting. For a more comprehensive view of neural networks, the reader can refer to Goodfellow, Bengio, and Courville (2016) which is freely available in electronic form.[4] To conduct the analysis in this paper, I use Tensorflow (Abadi et al., 2015) which is an open-source software commonly used in machine leraning applications, in particular neural networks.

In this paper, the class of neural networks I am using is so called deep feedforward networks (DFNs). For the rest of the paper, whenever I use term neural network, I refer to deep feedforward networks. DFNs are one of the simpler forms of neural networks and act as a basis for more complex neural network models. These models are called feedforward since the information flows through the network from input x, through the intermediate computations, to the output $\hat{y}$. The flow can be thought to be a series of functions. The intermediate computations between input and output are referred to as hidden layers. The first hidden layer would be $f^{(1)}(x)$, the second $f^{(2)}\left(f^{(1)}(x)\right)$ and so on. For instance, figure 1 depicts a simple DFN with two hidden layers. The first and the last layer are referred to as input and output layer respectively. Layers consist of multiple neurons or equivalently, nodes. The number of neurons defines the dimension of the vectors, which are often used to represent each of the layers.

[4] https://www.deeplearningbook.org/

The procedure which defines how the network computes output values from inputs is called forward propagation. For DFNs, it can be defined by the following set of computations. Let x be an input vector to the model and $\hat{y}$ the output. Moreover, $W^{(l)}$, $b^{(l)}$ and $g^{(l)}(\cdot)$ denote weight matrix, bias term and nonlinear activation function of l^{th} hidden layer respectively. Finally, let W, b and Θ be weight matrix, bias and activation function of an output layer. Forward propagation proceeds then as follows:

$$\begin{aligned} h^{(1)} &= g^{(1)}\left(W^{(1)}x + b^{(1)}\right) \\ h^{(2)} &= g^{(2)}\left(W^{(2)}h^{(1)} + b^{(2)}\right) \\ &\vdots \\ h^{(n)} &= g^{(n)}\left(W^{(n)}h^{(n-1)} + b^{(n)}\right) \\ \hat{y} &= \Theta\left(Wh^{(n)} + b\right) \end{aligned}$$

The output can hence be interpreted as function of the input, weights and biases:

$$\hat{y} = f\left(W^{(1)}, b^{(1)}, ..., W^{(n)}, b^{(n)}, W, b; x\right) \tag{1}$$

Note that without the nonlinear activation functions, network would be just performing subsequent affine transformations to the input vector. Introducing nonlinear activation functions allows network to learn complex nonlinear mappings between the input and output. Each of the activation functions are applied element wise whenever the preceding layer is of a dimension greater than 1. Throughout this paper, I use so called rectified linear unit (ReLU) activation function for all hidden layers. ReLU is defined by

$$g^{(l)}(z) = \max\{0, z\} \tag{2}$$

As I want the neural network to output probabilities, an activation function for the output layer needs to be chosen in a different way. One suitable function is Sigmoid function, which is monotone mapping of $z \in \mathbb{R}$ to the interval $(0, 1)$. Hence, the predictions can be interpreted as probabilities. Sigmoid function is defined as follows:

$$\Theta(z) = \frac{1}{1 + exp\,(-z)} \tag{3}$$

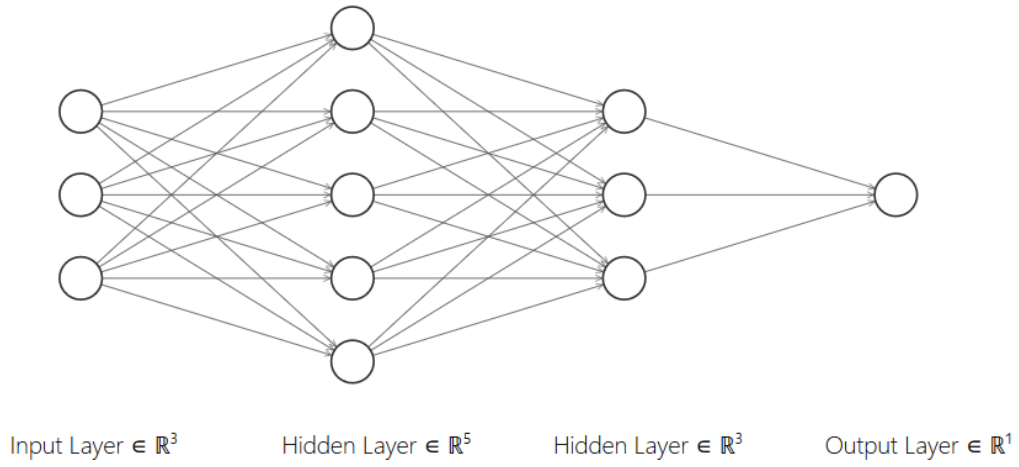
In this paper, I use neural network with four hidden layers, all having 30 neurons. Input layer is of a dimension 16 and output layer is one dimensional. The choice of number of hidden layers and layer depth is in line with the findings of Messmer (2017) and Gu et al. (2018) who document that on average, deeper networks perform better than shallower. Moreover, they find that higher number of neurons in the hidden layers gives better results.

Precisely, their finding suggest the use approximately more than 50 neurons. Since the number of features is lower in my paper, I use smaller number of neurons.

To express the number of parameters that need to be estimated in DFNs, let $n^{(l)}$ be the dimension of layer l. The dimension of $W^{(l)}$ is then $\mathbb{R}^{n^{(l)} \times n^{(l-1)}}$ and bias $b^{(l)}$ is vector of dimension $\mathbb{R}^{n^{(l)}}$. Hence, computing the combined number of elements in all $W^{(l)}$ and $b^{(l)}$ gives the number of model parameters. Take as an example the network in figure 1. Input layer is of dimension $\mathbb{R}^3$, the first hidden layer $\mathbb{R}^5$, second $\mathbb{R}^3$ and output $\mathbb{R}^1$. Total number of parameters is then

$$
\begin{aligned}
&(n^{(1)} \times n^{(input)}) + n^{(1)} + (n^{(2)} \times n^{(1)}) + n^{(2)} + (n^{(output)} \times n^{(2)}) + n^{(output)} \\
&= (5 \times 3) + 5 + (3 \times 5) + 3 + (1 \times 3) + 1 \\
&= 42
\end{aligned}
$$

Figure 1: This figure shows an example of deep feedforward nettwork. It has input layer of dimension $\mathbb{R}^3$, two hidden layers with dimensions of $\mathbb{R}^5$ and $\mathbb{R}^3$ and output layer with dimension $\mathbb{R}^1$



3.1 Loss function

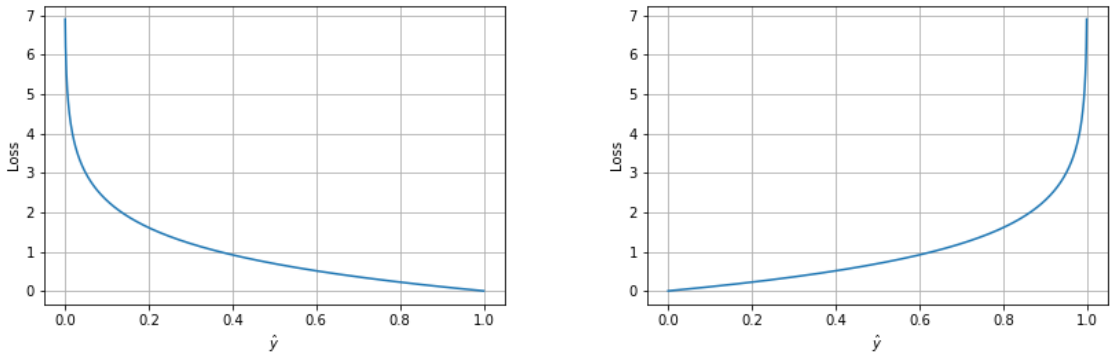
In general, parameter estimation in machine learning is done by minimizing some loss-function $L(\theta)$ with respect to θ, where θ denotes the set of model parameters. In the context of DFNs, set contains the weight matrices $W^{(l)}$ and bias vectors $b^{(l)}$. Given that I am working with a binary classification problem, i.e. $y_{i,t} \in \{0, 1\}$, I use so called cross-entropy loss function, which is an appropriate loss function whenever predicted values $\hat{y}_{i,t}$ are probabilities for an instance i at time t belonging to class 1. The cross-entropy loss

for binary classification is defined as

$$L(X, y, \theta; \phi) = -\frac{1}{N}\frac{1}{T}\sum_{i=1}^{N}\sum_{t=1}^{T}[y_{i,t}\log(\hat{y}_{i,t}) + (1 - y_{i,t})\log(1 - \hat{y}_{i,t})] \tag{4}$$

where ϕ is the set of model hyperparameters. Hyperparameters are parameters whose values are set before training the model.[5] The cross-entropy inside the summation is averaged over the number of time periods and firms in the data. Note that the contribution of a prediction $\hat{y}_{i,t}$ to the loss in (4) increases when the predicted probability diverges from the true label. Figure 2 illustrates the loss contribution of a term inside the summation of (4). This kind of loss-function ensures that the neural network is trained so that not only the prediction accuracy matters, but also the uncertainty of the predictions is taken into account. Prediction accuracy is defined as the proportion of labels that are classified correctly. A particular instance is classified to class 1 whenever its predicted probability $\hat{y}_{i,t}$ is greater than or equal to 0.5.

Figure 2: Left panel shows the loss contribution of one instance when true label $y_{i,t} = 1$ as a function of the predicted probability $\hat{y}_{i,t}$. Right panel shows the loss contribution when $y_{i,t} = 0$



3.2 Regularization

One of the main questions in machine learning is how to make a model that performs well on training data, but also on data it has not seen before, i.e. test data. Strategies which are designed to reduce prediction errors in test data, at the cost of performance on training data are called regularization techniques. This is necessary, since in theory the

[5]Take LASSO regression as an example, which is a regularized version of traditional OLS regression. LASSO-estimate is obtained by solving problem

$$\min_{\beta} ||y - X\beta||_2^2 + \lambda||\beta||_1$$

Here λ is hyperparameter which controls the degree of regularization. β is vector of model parameters. Note that when $\lambda = 0$ problem is equivalent to OLS regression.

loss in equation 4 could be made zero when the sufficient amount of model parameters are introduced, meaning that the model would learn the training data completely. However, such model will not perform well with data it has not seen before. This is an extreme example of overfitting and regularization is a weapon against it. The regularization techniques I use are L^2-penalty, dropout and early stopping. I describe each of the methods in the following separate subsections.

3.2.1 L^2-Regularization

In L^2-regularization, weight parameters in weight matrices $W^{(l)}$ are stacked into vector w. Usually, bias vectors are excluded from L^2-regularization as they affect directly only one layer, whereas weights interact between two layers, meaning that they are more prone to overfit (Goodfellow et al., 2016). In L^2-regularization, L^2-norm of the vector w is added to the loss function so the regularized loss is:[6]

$$\hat{L}(X, y, \theta; \phi) = -\frac{1}{N}\frac{1}{T}\sum_{i=1}^{N}\sum_{t=1}^{T}[y_{i,t}\log(\hat{y}_{i,t}) + (1 - y_{i,t})\log(1 - \hat{y}_{i,t})] + \frac{\lambda}{2}||w||_2^2 \qquad (5)$$

where hyperparameter λ is the regularization strength. The idea of L^2-regularization is to shrink components of the weight vector corresponding to the directions that do not contribute to reducing the loss function. That is, unimportant elements in w are forced closer to zero.

3.2.2 Dropout

Srivastava et al. (2014) provide a simple and computationally inexpensive regularization method for neural networks called dropout. During model training, each time data is passed through the network, all nonoutput neurons have probability q of being dropped out. Intuitive idea behind the dropout is that it forces the data adapt to a situation where all the neurons are not active. For instance, one neuron may learn some very specific pattern in the training data. When test data is passed through the network, it is unlikely that the model will generalize well. Dropout helps to avoid such situations. Srivastava et al. (2014) suggest the dropout rate for hidden layers should be smaller than 0.5 and for input layer smaller than 0.2. Note that dropout is only used in the training phase, not when the actual predictions are made. Essentially this means that each time training data is passed true the neural network, dropout rate is allowed to be larger than zero. Conversely, whenever model performance is evaluated with validation data or predictions are made with test data, dropout rates are set to zero.

[6] L^2-regularization is also known as Tikhonov regularization

3.2.3 Early stopping

Early stopping is a regularization method applied in the model training phase. The method proceeds as follows. During the training, model performance is evaluated in the distinct validation set every n^{th} trainning iteration and the model parameters are saved each time validation loss is decreased. Every time improvement happens, the count of evaluations in which validation performance has not improved is started and stored in variable j. Training is terminated when $j \geq p$ where p is called patience hyperparameter. After termination, the most recently saved model parameters are retrieved. Goodfellow et al. (2016) argue that the reason why early stopping acts as a regularizer is that it essentially restricts the "volume" of parameter space in a similar way as L^2 regularization does. Details of the early stopping algorithm are provided in the appendix.

3.3 Model training

Neural networks are typically trained so that data is passed to the training algorithm in so called mini-batches. This means that training data in each epoch is shuffled into distinct batches. Epoch means one iteration over the training data. Using mini-batches in model training is a trade-off between computational efficiency and final accuracy of the parameter estimates.

I use a mini-batch size of 128, which is consistent with Messmer (2017). For illustrative purposes, suppose that training data contains 10 000 instances and no early stopping is used. If number of epochs is set at 50, training is done with $\lfloor \frac{10000}{128} \rfloor \times 50 = 3900$ iterations.[7]

Training algorithms often rely on some modification of a gradient descent. Gradient based methods work so that the gradient of the loss function is evaluated at some initial point. Given that gradient is the direction where the function grows fastest, the initial point is updated by taking a step to the negative direction of the gradient. If α is the learning rate (or step size), gradient based methods are usually of the form

$$\theta_k = \theta_{k-1} - \alpha \nabla_\theta L(\theta)$$

Where ∇_θ denotes gradient with respect to the model parameters. The learning rate controls the step sizes the optimization algorithm takes. With too high learning rate, there is a risk of algorithm overshooting and missing the lowest point. In turn, too small rate may result in too slow convergence, or no convergence at all. Hence, I treat learning rate α as a trainable hyperparameter.

[7] $\lfloor \cdot \rfloor$ denotes floor-operator

In mini-batch training, the computed gradient is not obviously the true gradient, but an estimate based on the gradient evaluated with small subset of the whole training data, giving the mini-batch training a stochastic nature. This motivates the use of Adaptive moment estimation (ADAM) algorithm by Kingma and Ba (2014). ADAM is an extension to stochastic gradient descent (SGD) algorithm. In ADAM, gradient of the loss function is treated as a random variable and exponential weighted moving average of estimated gradients is used at each iteartion, such that the latest gradient estimate has the highest weight. ADAM also allows for adaptive learning rate for each of the components in the gradient. This prevents the problems arising when gradient is of a different magnitude of its components. I treat the learning rate of ADAM as a hyperparameter. Details of the ADAM implementation is given in appendix.

3.4 Hyperparameter optimization

As earlier, I denote the set of hyperparameters by $\phi := \{\lambda, \alpha, q_h, q_i, p\}$. Here, λ is the L^2 regularization strength, α learning rate for optimization algorithm, q_h and q_i dropout rate for input and output layers respectively and p is the patience parameter. Hyperparameters are optimized so that first the neural network is trained with training data. Then, model performance is evaluated in a distinct validation set and the set of hyperparameters that gives the lowest validation loss is chosen and is used to make the final predictions with test data. I describe the data division into training, validation and test sets in the following section.

Formally, hyperparameter optimization proceeds as follows. Let $\phi_i, \quad i = 1, ..., n$ be a particular set of hyperparamters. Then, neural network is trained for each ϕ_i, i.e. n times, by minimizing regularized loss in (5):

$$\hat{\theta}_i = \underset{\theta}{\operatorname{argmin}} \hat{L}(X_{train}, y_{train}, \theta; \phi_i)$$

Optimal model parameters and hyperparameters are then obtained by

$$\hat{\theta}, \hat{\phi} = \underset{i=1,...,n}{\operatorname{argmin}} \hat{L}(X_{val}, y_{val}, \hat{\theta}_i; \phi_i) \tag{6}$$

An optimal way to find hyperparameters would of course be to do a grid search over the whole space of the hyperparameter values, but when the number of hyperparameters increases, this approach would become computaionally infeasible very quickly. Suppose that I would divide each dimension in the hyperparameter space into 10 grid points and then do a grid search. This means going through every possible hyperparameter combination in the discretized space. In this case and given the five hyperparameters, the

model would have to be trained $10 \times 10 \times 10 \times 10 \times 10 = 100000$ times, even with a such sparse choice of grids. This problem is usually referred to as curse of dimensionality.

To tackle the curse of dimensionality, I conduct so called randomized search of hyperparameters. This means that I randomnly pick uniformly the value for each hyperparameter from a predefined interval. I do this 200 times to obtain 200 random sets of hyperparameters. Intuitively, grid search wastes computational resources moving along insensitive directions. Randomized search tries to overcome this problem by assuming that the hyperparameter space is covered well enough by randomly drawing a smaller number of hyperparameter sets. This number is smaller than the number of required grid search iterations would be. The intervals for the hyperparameters are as follows:

$$\begin{aligned} \lambda &\in \left[10^{-7}, 10^{-2}\right] \\ \alpha &\in \left[10^{-6}, 10^{-2}\right] \\ q_h &\in [0, 0.5] \\ q_i &\in [0, 0.2] \\ p &\in \{20, ..., 100\} \end{aligned}$$

Interval choices are motivated by industry standards[8] and related research, e.g. Messmer (2017).

4 Data and methodology

This section describes the data used in my analysis and how I construct the features which act as inputs to the neural networks. I also demonstrate how the data is divided into training, validation and test sets and the way the neural network cross-sectional momentum strategy is constructed. I use daily and monthly stock price data from year 1948 to 2012 which is obtained from The Center for Research in Security Prices (CRSP). I restrict my analysis to common stocks (CRSP share code 10 or 11) trading in NYSE, Nasdaq and AMEX with share price above \$1. The risk factors used in spanning tests, size, value and short-term reversal, are obtained from Kenneth French's database.[9]

[8] For instance, Kingma and Ba (2014) suggest default learning rate of 10^{-3}, which is also the default value in Tensorflow Python library

[9] http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/

4.1 Feature construction

In the context of machine learning, variables used as an input to the model are usually referred to as features and the predicted variables as labels. The feature set I use consists of two categories: momentum signals, i.e. past returns from different time spans, and stock level risk indicators. Label in turn will be dummy variable indicating whether stock i's return is above the cross-sectional median in month t.

4.1.1 Momentum signals

For momentum signals, I use the definition consistent with the literature, see e.g. Daniel and Moskowitz (2016). In particular, let K be a look back period in months. Then the momentum signal for stock i in month t is the cumulative return between months $t-2$ and $t-K$:

$$MOM_{i,t,K} := \prod_{s=t-2}^{t-K} (1 + r_{i,s}) - 1 \tag{7}$$

Here, $r_{i,t}$ denotes monthly return on stock i on month t and $K \in \{2, 3, ..., 12\}$. Note that the most recent month is excluded to avoid the short-term reversal effect shown for instance by Jegadeesh (1990) and Lehmann (1990).

4.1.2 Volatility measures

Motivated by the literature which suggests that volatility scaled momentum strategies outperform the benchmark strategy, I use simple ex-ante volatility estimates as my first set of firm level risk indicators. I compute three ex-ante volatility estimates from daily returns by using rolling standard deviation with estimation windows of 21, 126 and 252 days. I use such specification since these windows represent the average number of trading days within a month, 6 months and 12 months respectively. Volatility estimates for stock i at time t are denoted by $\hat{\sigma}_{i,t}^{(21)}$, $\hat{\sigma}_{i,t}^{(126)}$ and $\hat{\sigma}_{i,t}^{(252)}$

4.1.3 Tail dependence measures

In addition, I include estimates of the lower and upper tail dependence with the market returns for each stock. Lower (uppper) tail dependence measure can be interpreted as a probability that stock i's return is below (above) quantile q given that the contemporaneous market return is below (above) the same quantile q. Lower tail dependence (LTD)

measure is obtained when q is let to approach to zero from right whereas UTD measeure is obtained when q approaches one from left.

The idea behind the inclusion of lower and upper tail dependence measures is to help the neural network to predict the rebound effect, that is often behind momentum crashes. As Daniel and Moskowitz (2016) document, momentum crashes happen usually after bear market when market rebounds. I expect that having an estimate of the individual stock's dependency with extreme market movements could help the neural network to filter out some of the stocks in the short leg of the strategy that drive the momentum crashes. Common characteristics for such stocks are firstly, tendency to fall during the market crashes a lot and ending up in the short leg, and secondly, tendency to rebound with market. I assume that LTD and UTD measures are proxies of these characteristics. Moreover, Ruenzi and Weigert (2018) show that crash factor constructed using same LTD measure I am considering, explains a large part of momentum strategy's abnormal returns.

Specifiaction of LTD and UTD mesures follows the one by Chabi-Yo et al. (2018). They develop a copula based approach, where they find the best fitting copula for individual stock's and market's daily returns from a set of different parametric copulas and their combinations. Copulas are probability distributions which encode the dependence structure of two or more random variables. When it comes to modeling tail behavior of random variables, copulas have several desirable properties. First, they do not assume anything about the parametric form of the joint distribution between random variables. Secondly, copula parameters can be estimated from data coming from much narrower time window than other lower tail risk measures, such as Hill-estimator by Hill (1975), which uses only observations below some predefined quantile. And last, it is possible to express LTD and UTD measures in an analytic form for a particular set of copulas.[10]

Formally, LTD measure is defined as follows. Let r_i denote individual stock's daily return and r_m daily return on market. Also, define $F_{r_i}(x)$ and $F_m(x)$ as the empirical marginal distributions of r_i and r_m respectively. Then lower tail dependence is defined as follows:

$$p_i^L(q) := P\left[r_i \leq F_{r_i}^{-1}(q) | r_m \leq F_{r_m}^{-1}(q)\right] \tag{8}$$

$$LTD_i := \lim_{q \to 0^+} p_i^L(q) \tag{9}$$

Similarly, UTD tail dependence measure is defined as

$$p_i^U(q) := P\left[r_i \geq F_{r_i}^{-1}(q) | r_m \geq F_{r_m}^{-1}(q)\right] \tag{10}$$

[10]Hence the set of copulas I use is limited to those whose LTD and UTD measures are available in an analytical form

$$UTD_i := \lim_{q \to 1^-} p_i^U(q) \tag{11}$$

Tail dependence measures are estimated such that at the end of month $t-1$ daily returns of stock i and market from past 12 months are used to fit the copula. Then, LTD and UTD measures are computed analytically from the estimated copula for month t.

After feature construction, I pool all features over time and firms into a design matrix X and label vector y. Hence, rows of X are vectors of the form

$$\begin{bmatrix} MOM_{i,t,2} & \dots & MOM_{i,t,12} & \hat{\sigma}_{i,t}^{(21)} & \hat{\sigma}_{i,t}^{(126)} & \hat{\sigma}_{i,t}^{(252)} & LTD_{i,t} & UTD_{i,t} \end{bmatrix}$$

I also scale all features cross-sectionally so that the cross-sectional mean and variance in month t are 0 and 1 respectively for each feature. This is done to make data comparable in time dimension such that the cross-sectional variation is preserved.

4.2 Training, validation and test sets

In machine learning, data is usually divided into three distinct sets, training, validation and testing sets. The first, training set, is used to train the model, which essentially means estimation of model parameters. This is done by minimizing some predefined loss function with respect to the model parameters by optimization algorithm. As decribed in the previous sections, I use so called reguralized cross-entropy loss function. In addition to the model parameters, hyperparameters need to be specified. Hyperparameters are chosen by first training the model on the training set and then evaluating the model's performance on distinct validation set. The set of hyperparameters that produces the lowest loss on the validation set is selected. Then, the best performing model is used to make predictions on the test set. This kind of procedure ensures that the predictions are made on data that the model has not seen before.

Given that I am using panel data, I will use rolling estimation window to maintain the chronological ordering of the data: in year k, I take data from years $k-1, k-2, ..., k-16$ and use the most recent 4 years as the validation set and the rest as the training set. After choosing the best model parameters and hyperparameters, I make predictions for the next 5 years, i.e. years $k, k+1.k+2, k+3, k+4$. Next, I move the window five years forward. This exercise is repeated through years 1964 and 2012.[11] Hence, the first year in the data is 1948 given the window of 16 years. Table 1 illustrates the division of data into training, validation and testings sets on a rolling basis. The amount of available

[11] Also Barroso and Santa-Clara (2015) and Daniel and Moskowitz (2016) use estimation window that ends in December 2012

computational resources is the key driver of the decision to train model only every fifth year.

Table 1: An illustration of rolling split of data into training, validation and testing sets. Training set is used to fit model parameters and validation set to find optimal set of hyperparamters. Testing set is the period for which predictions are made.

Testing set	1964-1968	1969-1973	1974-1978	...	2004-2008	2009-2012
Validation set	1960-1963	1965-1968	1970-1973	...	2001-2004	2005-2008
Training set	1948-1959	1953-1964	1958-1969	...	1989-2000	1993-2004

4.3 Portfolio construction

4.3.1 Benchmark momentum

Benchmark momentum strategy is formed in a consistent way with Daniel and Moskowitz (2016). At the end of month t, sort stocks into decile portfolios based on their 12-month momentum signal, that is, cumulative return during period from $t-12$ to $t-2$. Signal is computed as in equation (7) such that $K = 12$. The first (last) portfolio consists of stocks with the lowest (highest) 12-month momentum signal. Note that I skip the most recent month due to the short term reversal effect.

4.3.2 Neural network enhanced momentum

Neural network enhanced momentum portfolios are constructed in a slightly different manner. Given that I am studying momentum as a cross-sectional phenomenon, I use similar approach as Takeuchi and Lee (2013). Their approach is to treat portfolio construction as a binary classification problem where the neural network is trained to predict the value of $y_{i,t+1} \in \{0, 1\}$, where $y_{i,t+1}$ is an indicator whether particular stock's return is above the cross-sectional median during month $t+1$. In the context of classification problems, machine learning models are often designed such that they output a probability of an instance to belonging into each class. I will hence construct my neural networks such that they predict the probabilities

$$P\left(y_{i,t+1} = 1\right)$$

Then, I use the predicted probabilities to sort stocks into decile portfolios at the start of each month so that the first (last) portfolio consists of stocks with the highest (lowest) estimated probability of having a return that is above (below) the cross-sectional median. Intuitively, $P\left(y_{i,t+1} = 1\right)$ can be interpreted as a signal of the predicted strength of relative performance.

5 Results

I start the analysis of the results by presenting statistics of the prediction accuracy of the trained neural network. Also, the section summarizes neural network enhanced cross-sectional momentum strategy returns and compares it to the benchmark momentum strategy. Moreover, I test strategy's loadings on traditional risk factors and analyze the risk profile of the strategy.

5.1 Neural network prediction accuracy

One way to demonstrate the performance of a classifier is a confusion matrix. Confusion matrix has two dimensions: predicted labels as columns and true labels as rows. Then, each element of the matrix represents proportions of how accurately each label was classified. In my case, labels are just "winner" and "loser". Being winner is as defined earlier, i.e., stock having a return above the cross-sectional median during month t is considered as a winner.

Figure 3 shows the normalized confusion matrix for the winner-loser classification problem. Normalization ensures that the row sums equal 1. All numbers are computed using predictions from Januray 1964 to December 2012. The top left corner shows the proportion of which of the stocks predicted as losers were actually losers. Hence, the number 0.46 indicates that each time the model predicts that a stock is a loser, it is correct 46% of the time. The bottom right corner shows the same number but for winners, 0.63. In turn, the total accuracy of the neural network classifier is 0.54. The total accuracy is just proportion of correctly classified instances. These numbers indicate that the neural network is able to predict the winners more accurately, on average. However, since the actual investment strategy is based on the decile portfolios, and in particular, the extreme deciles, prediction accuracy in the first and last decile would be of more interest.

Table 2 reports prediction accuracy within each of the ten decile portfolios. The process to construct the portfolios was described in the previous section. It actually seems that the prediction accuracy is at the highest (60.22%) in the first decile portfolio, which consists of the stock which the neural network is the most "sure" to be losers. Not surprisingly, prediction accuracy is also high in the last portfolio, 57.48%. These numbers indicate that although the neural network on average classifies winners with higher accuracy, within the extreme deciles it does better job with losers.

Figure 3: Confusion matrix of the predictions. Numbers are normalized such that the row sums equal 1.

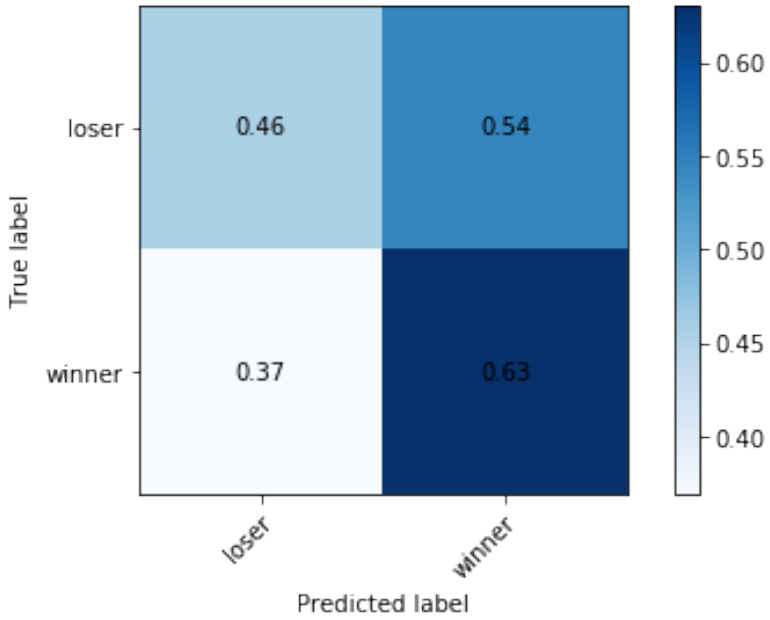


Table 2: Prediction accuracy within the decile portfolios. Decile portfolios are those formed in the neural network portfolio construction phase.

Decile	1	2	3	4	5	6	7	8	9	10
Accuracy	60.22%	55.65%	54.05%	50.65%	50.66%	52.05%	53.45%	54.74	55.88%	57.48%

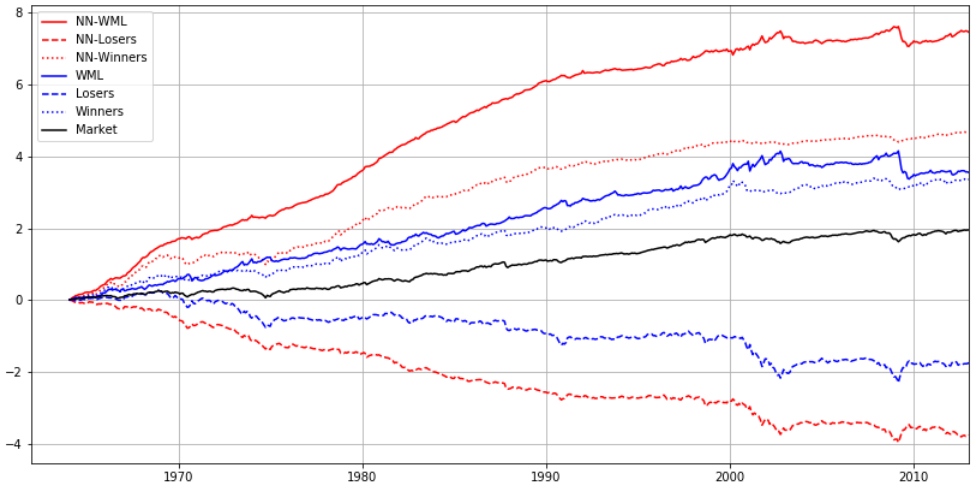
5.2 Strategy performance

Figure 4 shows cumulative value weighted returns for neural network and benchmark momentum portfolios and cumulative return for CRSP value weighted market portfolio in time window from January 1964 to December 2012. In particular, time series for loser, winner and winner-loser (WML) long-short portfolios are reported. Abbreviation "NN" in the figure 4 stands for neural network and is used to distinguish neural network momentum from benchmark momentum strategy. All returns are reported on logarithmic scale with a base of 10. This means for example that a hypothetical $1 investment at the start of January 1964 in NN-WML portfolio would have grown to approximately to $28 000 000. The corresponding number for WML portfolio is approximately $ 3 500.

To compare the strategy performance during the months when the benchmark momentum has been documented to suffer severe losses, table 3 reports the worst 15 months for the benchmark strategy with contemporaneous month the NN-WML and market return. Clearly, it is observable that the NN-WML portfolio does not suffer from as severe losses as

the WML portfolio.[12] Corresponding month market return is also included to highlight the rebound effect, i.e. during market rebounds, short leg often consists of high-beta stocks generating adverse losses. Average returns for the WML, NN-WML and market during the 15 months in table 3 are -28.31%, -16.34% and 5.47% respectively.

Figure 4: Cumulative returns for winners, losers and winner-loser portfolios for benchmark and neural network enhanced cross-sectional momentum strategies. Also, cumulative return for CRSP value weighted market portfolio is included. I use abbreviations WML and NN-WML for the winner-minus loser portfolios, where NN stands for neural network. Returns are plotted in $\log_{10}$ scale. Time period is January 1964 - December 2012. All portfolios are value weighted.



[12] As Daniel and Moskowitz (2016) document, momentum strategy suffers from even more extreme losses during the Great recession in 1930s: in July and August of 1932 the WML portfolio had returns of -74% and 61% respectively.

Table 3: The worst fifteen months for WML portfolio with contemporaneous month NN-WML and market returns. All numbers are reported as percentages.

Month	WML	NN-WML	vwretd
2001-01	-46.51	-20.40	4.19
2009-04	-46.10	-34.96	10.94
2009-03	-42.33	-26.55	8.70
2002-11	-35.49	-27.08	6.08
2009-08	-30.68	-18.01	3.13
2001-11	-25.51	-16.41	7.88
1991-02	-24.62	-5.97	7.59
2001-10	-24.20	-20.82	2.79
1970-09	-23.50	-10.44	4.77
2008-01	-21.76	-6.29	-6.21
1974-01	-21.72	-11.85	0.47
2009-05	-21.35	-20.58	6.78
2012-01	-21.00	-17.89	5.41
1975-01	-20.10	-3.57	13.90
1973-07	-19.78	-4.41	5.69

Table 4: Summary statistics for benchmark momentum and neural network based cross-sectional momentum strategies. The time period is January 1964 - December 2012. Numbers are reported in percentages and in monthly basis, except for Sharpe ratio which is annualized. First ten columns of panels A and B represent decile portfolios where column 1 (10) consists of stocks whose momentum signal is lowest (highest). Last columns, WML and NN-WML, are a zero-investment long-short portfolio buying portfolio 10 and selling portfolio 1. All portfolios are value weighted.

Panel A: Summary statistics for benchmark momentum strategy returns

	1	2	3	4	5	6	7	8	9	10	WML
Mean return	-0.23	0.30	0.48	0.78	0.80	0.80	0.94	1.10	1.18	1.56	1.79
t-statistic	-0.58	0.97	1.86	3.52	3.96	4.27	5.03	5.67	5.43	5.50	5.12
σ	9.54	7.41	6.31	5.36	4.89	4.56	4.55	4.72	5.26	6.89	8.49
Sharpe ratio (ann.)	-0.08	0.14	0.27	0.50	0.57	0.61	0.72	0.81	0.78	0.79	0.73
Median	-0.28	0.35	0.57	0.66	0.84	0.86	1.29	1.30	1.75	1.73	2.31
Min	-34.47	-27.56	-26.97	-22.77	-19.36	-20.30	-22.49	-21.54	-26.50	-27.25	-46.51
Max	47.42	36.32	34.14	26.61	20.83	15.76	18.57	18.78	22.40	34.87	34.24
2%-percentile	-20.28	-17.40	-13.79	-11.06	-10.82	-9.22	-8.53	-8.57	-9.63	-12.90	-21.09
5%-percentile	-14.94	-11.48	-8.92	-7.27	-7.32	-7.12	-6.36	-6.69	-8.18	-10.01	-13.53

Panel B: Summary statistics for neural network momentum strategy returns.

	1	2	3	4	5	6	7	8	9	10	WML
Mean return	-1.16	-0.16	0.23	0.52	0.58	0.85	0.90	1.11	1.27	2.02	3.18
t-statistic	-3.64	-0.53	0.81	1.97	2.54	4.01	4.67	5.84	6.27	8.24	11.81
σ	7.72	7.40	6.88	6.41	5.57	5.13	4.69	4.60	4.92	5.95	6.53
Sharpe ratio (ann.)	-0.52	-0.08	0.12	0.28	0.36	0.57	0.67	0.83	0.90	1.18	1.69
Median	-1.28	-0.03	0.20	0.57	0.81	1.12	1.13	1.34	1.58	1.93	3.41
Min	-28.11	-36.20	-27.69	-28.70	-24.94	-25.56	-21.39	-22.10	-22.63	-23.78	-34.96
Max	36.28	30.62	27.13	35.21	21.20	21.15	17.09	19.02	21.77	22.25	26.73
2%-percentile	-19.16	-16.60	-17.57	-13.76	-14.10	-10.93	-10.47	-8.66	-8.90	-11.70	-15.58
5%-percentile	-12.81	-11.79	-10.15	-9.13	-8.83	-7.79	-6.86	-6.65	-6.60	-7.27	-6.68

Table 4 reports summary statistics for value weighted benchmark and neural network enhanced cross-sectional momentum strategies. Corresponding equal weighted statistics are reported in the appendix. On the basis of mean return and sharpe ratio, neural network enhanced momentum strategy outperforms benchmark momentum strategy by a large margin: WML and NN-WML have monthly mean returns of 1.79% and 3.18% and annualized Sharpe ratios of 0.73 and 1.69 respectively. That is, based on Sharpe ratio, neural networks more than doubles the performance. Moreover, comparing NN-WML returns to dynamic weigting strategy by Daniel and Moskowitz (2016), which yields annual Sharpe ratio of 1.194, neural network strategy still prevails.

Comparing mean returns of the ten decile portfolios suggests that the biggest improvement the use of neural network give is that it is able to predict the future losers more accurately than benchmark momentum strategy. This is also in line with the numbers in the table 2. The loser deciles have mean returns of -1.16% and -0.23% (difference of 0.93%-points) respectively, meaning that shorting these portfolios has on average positive effect on both strategies. In general, it seems that neural network does better job at predicting a particular stock ending in the extreme deciles as differences of mean returns are lower in the mid deciles.

Moreover, the summary statistics tell also the same story as Table 3: neural network is also able to reduce the severity of the extreme low returns of the benchmark momentum strategy. Benchmark momentum's 2%-percentile return of -21.09% is increased to -15.58%. Similarly, 5%-percentile return is improved from -13.53% to -6.68%, suggesting that the left tail of NN-WML returns is lighter than WML returns'. I will provide further analysis of the risk-management properties of the neural network in the upcoming sections.

5.3 Spanning tests

Table 5: This table reports regression results for 7 different regression settings with NN-WML as the dependent variable. Regressions are estimated using monthly returns from January 1964 to December 2012. The first row is the estimated intercept term in the regression and Mkt-RF stands for excess market return. SMB and HML are size and value factors by Fama and French (1993), WML is the becnhmark momentum portfolio, CRASH is the market wide crash risk factor by (Chabi-Yo et al., 2018) and ST-REV is the short-term reversal factor. Numbers reported are the regression coefficients and the standard errors in parentheses. All standard errors are estimated using Newey and West (1986) method with 2 monthly lags. ***, ** and * indicate statistical significance at 1%, 5% and 10% levels respectively.

	Dependent variable: NN-WML						
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Intercept	0.033***	0.034***	0.023***	0.03***	0.031***	0.023***	0.023***
	(0.003)	(0.003)	(0.003)	(0.003)	(0.003)	(0.003)	(0.003)
Mkt-RF	-0.273***	-0.141	0.07	-0.217**	-0.177	0.052	0.038
	(0.096)	(0.102)	(0.065)	(0.109)	(0.119)	(0.072)	(0.071)
SMB		-0.594***	-0.56***	-0.545***	-0.527***	-0.553***	-0.562***
		(0.127)	(0.171)	(0.143)	(0.156)	(0.176)	(0.176)
HML		0.049	0.296***	0.269*	0.285*	0.323***	0.317***
		(0.156)	(0.108)	(0.147)	(0.146)	(0.105)	(0.102)
WML			0.47***			0.456***	0.466***
			(0.051)			(0.047)	(0.047)
CRASH				0.813***	0.803***	0.126	0.116
				(0.195)	(0.2)	(0.124)	(0.127)
ST-REV					-0.194		0.099
					(0.135)		(0.123)
Observations	588.0	588.0	588.0	588.0	588.0	588.0	588.0
Adjusted R2	0.034	0.107	0.456	0.183	0.19	0.457	0.458
Residual Std. Error	0.064	0.062	0.048	0.059	0.059	0.048	0.048
Note:							*p<0.1; **p<0.05; ***p<0.01

Table 6: This table reports regression results for 7 different regression settings with WML as the dependent variable. Regressions are estimated using monthly returns from January 1964 to December 2012. The first row is the estimated intercept term in the regression and Mkt-RF stands for excess market return. SMB and HML are size and value factors by Fama and French (1993), NN-WML is the neural network momentum portfolio, CRASH is the market wide crash risk factor by (Chabi-Yo et al., 2018) and ST-REV is the short-term reversal factor. Numbers reported are the regression coefficients and the standard errors in parentheses. All errors are estimated using Newey and West (1986) method with 2 monthly lags. ***, ** and * indicate statistical significance at 1%, 5% and 10% levels respectively.

	Dependent variable: WML						
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Intercept	0.02***	0.022***	-0.004	0.016***	0.018***	-0.005	-0.003
	(0.003)	(0.003)	(0.005)	(0.003)	(0.003)	(0.004)	(0.004)
Mkt-RF	-0.373***	-0.449***	-0.335***	-0.589***	-0.459***	-0.439***	-0.351***
	(0.123)	(0.127)	(0.094)	(0.118)	(0.132)	(0.093)	(0.084)
SMB		-0.073	0.435*	0.017	0.075	0.432**	0.456**
		(0.224)	(0.236)	(0.182)	(0.147)	(0.203)	(0.18)
HML		-0.526**	-0.565***	-0.118	-0.068	-0.296*	-0.251*
		(0.246)	(0.187)	(0.224)	(0.211)	(0.158)	(0.147)
NN-WML			0.784***			0.689***	0.659***
			(0.084)			(0.088)	(0.088)
CRASH				1.505***	1.473***	0.975***	0.975***
				(0.215)	(0.229)	(0.197)	(0.19)
ST-REV					-0.629***		-0.46***
					(0.168)		(0.155)
Observations	588.0	588.0	588.0	588.0	588.0	588.0	588.0
Adjusted R2	0.038	0.063	0.398	0.219	0.268	0.458	0.484
Residual Std. Error	0.083	0.082	0.066	0.075	0.073	0.062	0.061
Note:							*p<0.1; **p<0.05; ***p<0.01

To investigate more formally the success of the neural network momentum strategy, I conduct various spanning tests with respect to the benchmark momentum stratrgy and other risk factors. Precisely, I run regressions of the following form:

$$\text{NN-WML}_t = \beta^T_{\text{NN-WML}} f_t + \varepsilon_{\text{NN-WML},t} \tag{12}$$

$$\text{WML}_t = \beta^T_{\text{WML}} f_t + \varepsilon_{\text{WML},t} \tag{13}$$

Where f_t denotes a vector of selected factors and $\varepsilon_{i,t}$ is the error term. Table 5 shows results of seven time-series regressions with the neural network enhanced momentum portfolio (NN-WML) as a dependent variable. Each column represents a different regression

setup. Variable Mkt-RF stands for excess market return, SMB and HML are size and value factors by Fama and French (1993), WML is benchmark momentum portfolio, CRASH is market-wide crash factor by Chabi-Yo et al. (2018) and ST-REV is short-term reversal factor. Value inside parentheses is the standard error of the estimator. In turn, table 6 reports the results from the same set of regressions, but with the WML as a dependent variable. I include benchmark momentum as a regressor to the NN-WML portfolio to ensure that neural network actually forms a momentum strategy. Given the positive and significant regression coefficients to the WML, NN-WML actually has a strong loading to the benchmark momentum. Also, adding the NN-WML as a regressor to the WML regressions, results in insignificant alphas as can be seen from table 6 indicating that neural network enhanced momentum strategy's returns are not captured by benchmark momentum strategy.

In all of the regression setups in table 5, the NN-WML portfolio alpha is positive and significant at 1%-level, ranging from 2.3% to 3.4% per month. It is also of a higher magnitude than for the WML portfolio in all regression setups. The WML portfolio has positive and significant alphas for regression setups without including the NN-WML as an explanatory variable. This provides, not surprisingly, more evidence that neural network is able to enhance the traditional cross-sectional momentum strategy.

Considering loadings to the CRASH factor, the WML portfolio has a positive and significant loading in regressions (4)-(7). For the NN-WML, the loading is positive and significant in regressions (4) and (5), but approximately half of the magnitude of the WML portfolio's loadings, indicating that the NN-WML has exposure to market wide crash risk but, however, the neural network enhanced momentum strategy is able to reduce the crash risk compared to the benchmark momentum strategy.

5.4 Sensitivity to size filters

Somewhat surprisingly, NN-WML has a strong negative and statistically significant loading on size factor (SMB). Conversely, benchmark momentum does not have significant loading on the size factor in any of the regression setups. This means that the performance of the neural network momentum strategy might partly be driven by the shorting of small cap stocks, which may be unrealistic if applied in real markets. To investigate this observation further, I evaluate the performances of both momentum strategies using reduced datasets. I use stocks whose market capitalization is above the 1) 25%, 2) 50% and 3) 75% percentile of all firms. Precisely, each time the momentum portfolios are re-balanced, I only consider stocks fulfilling the market capitalization criteria on a cross-sectional basis. Note that I do not re-train the neural network for the different cases, but use the model that was trained using the whole dataset. This means that statistics for

the neural network momentum strategy most likely look worse than in a situation where network would have been trained under the assumption that investing is possible only on firms filling a predefined size criteria.

Tables 7, 8 and 9 report mean return, t-stat of the mean and Sharpe ratio for each of the three cases. For each subset, the neural network momentum outperforms the benchmark momentum on the basis of mean return and Sharpe ratio. However, the neural network momentum returns are much more sensitive to the size filters, indicating that the neural network seems to capture some characteristics that the benchmark momentum does not exhibit. These results indicate that short positions in small market capitalized firms are at least to some extent driving the strong performance. This may overestimate the performance if applied in practice, since short selling is not always possible. Restrictions in short selling most often are a concern of small cap stocks, see e.g. Duan, Hu, and McLean (2010). However, the long leg, i.e. bet on the past winners, of the neural network momentum strategy is able to outperform the benchmark long-short strategy on the basis of Sharpe ratio in all settings. This means that despite of the possible limitations arising from the assumed possibility of short selling, I deduce that the neural network momentum strategy manages to outperformn the benchmark strategy by a relatively large margin even after limiting stock universe to contain only larger firms.

Table 7: This table and tables 8 and 9 report the benchmark and neural network momentum strategy returns after considering only firms having market capitalization above the 25%, 50% and 75% cross-sectional percentiles.

Panel A: Benchmark momentum strategy returns, firms with market capitalization above 25%-percentile

	1	2	3	4	5	6	7	8	9	10	WML
Mean return	-0.33	0.30	0.46	0.77	0.79	0.77	0.95	1.08	1.19	1.58	1.91
t-stat	-0.82	0.95	1.78	3.51	3.89	4.08	5.03	5.56	5.49	5.50	5.28
Sharpe ratio (ann.)	-0.12	0.14	0.25	0.50	0.56	0.58	0.72	0.79	0.78	0.79	0.75

Panel B: Neural network momentum strategy returns, firms with market capitalization above 25%-percentile

	1	2	3	4	5	6	7	8	9	10	WML
Mean return	-1.04	-0.15	0.22	0.51	0.58	0.84	0.90	1.10	1.26	1.97	3.01
t-stat	-3.22	-0.48	0.78	1.94	2.51	3.99	4.66	5.82	6.24	8.08	11.07
Sharpe ratio (ann.)	-0.46	-0.07	0.11	0.28	0.36	0.57	0.67	0.83	0.89	1.15	1.58

Table 8

Panel A: Benchmark momentum strategy returns, firms with market capitalization above 50%-percentile

	1	2	3	4	5	6	7	8	9	10	WML
Mean return	-0.30	0.31	0.46	0.77	0.79	0.76	0.94	1.07	1.17	1.56	1.86
t-stat	-0.73	0.98	1.76	3.51	3.88	4.04	4.99	5.51	5.42	5.44	5.02
Sharpe ratio (ann.)	-0.10	0.14	0.25	0.50	0.55	0.58	0.71	0.79	0.77	0.78	0.72

Panel B: Neural network momentum strategy returns, firms with market capitalization above 50%-percentile

	1	2	3	4	5	6	7	8	9	10	WML
Mean return	-0.72	-0.10	0.23	0.51	0.57	0.84	0.89	1.09	1.24	1.87	2.58
t-stat	-2.17	-0.34	0.79	1.91	2.48	3.96	4.62	5.77	6.14	7.76	9.23
Sharpe ratio (ann.)	-0.31	-0.05	0.11	0.27	0.35	0.57	0.66	0.82	0.88	1.11	1.32

Table 9

Panel A: Benchmark momentum strategy returns, firms with market capitalization above 75%-percentile

	1	2	3	4	5	6	7	8	9	10	WML
Mean return	-0.24	0.31	0.43	0.76	0.78	0.75	0.93	1.05	1.14	1.53	1.76
t-stat	-0.54	0.97	1.63	3.43	3.81	3.98	4.96	5.41	5.24	5.26	4.36
Sharpe ratio (ann.)	-0.08	0.14	0.23	0.49	0.55	0.57	0.71	0.77	0.75	0.75	0.62

Panel B: Neural network momentum strategy returns, firms with market capitalization above 75%-percentile

	1	2	3	4	5	6	7	8	9	10	WML
Mean return	-0.32	-0.07	0.24	0.50	0.55	0.82	0.87	1.06	1.19	1.69	2.01
t-stat	-0.93	-0.23	0.84	1.85	2.37	3.88	4.51	5.61	5.91	7.14	6.57
Sharpe ratio (ann.)	-0.13	-0.03	0.12	0.26	0.34	0.55	0.64	0.80	0.84	1.02	0.94

5.5 Factor-like formulation of neural network enhanced momentum

To study more the effect of size in the neural network momentum portfolio, I construct a double sorted factor portfolio. I follow the widely used approach by Fama and French (1993), where the effect of size is filtered out by first dividing stocks into two size categories so that the market capitalization breakpoint is the median of NYSE stocks.[13] To be precise, each month, I take a size median of the all stocks listed in NYSE and then I allocate all stocks into two categories, small and big. Next, within these size categories, I use the predicted probabilities of being a winner and allocate stocks into three groups

[13]Kenneth French provides time series of NYSE breakpoints in his personal website: mba.tuck.dartmouth.edu/pages/faculty/ken.french/

within the size categories. The three groups are formulated so that stocks that have predicted probability of being a winner below 30% decile are labeled as losers and those with predicted probability of being a winner above 70% decile are labeled as winners. The NN-WML factor is formulated then as

$$factor_t = \frac{1}{2}(SW + BW) - \frac{1}{2}(SL + BL) \tag{14}$$

where S (B) stands for category small (big) and W (L) for winner (loser), as before. Table 10 reports summary statistics for NN-WML factor.

Table 10: This table shows the summary statistics for factor-like formulation of the neural network momentum strategy. I follow the approach by Fama and French (1993) and each month sort the stocks into two size portfolios based on the NYSE deciles and then within these portfolios, allocate the stocks into three groups based on the predicted probability of being a winner in the upcoming month. S, B, W and L stand for small, big, winner and loser respectively. Column factor is as in equation (14) Reported numbers are percentages except for Sharpe ratio which is annualized.

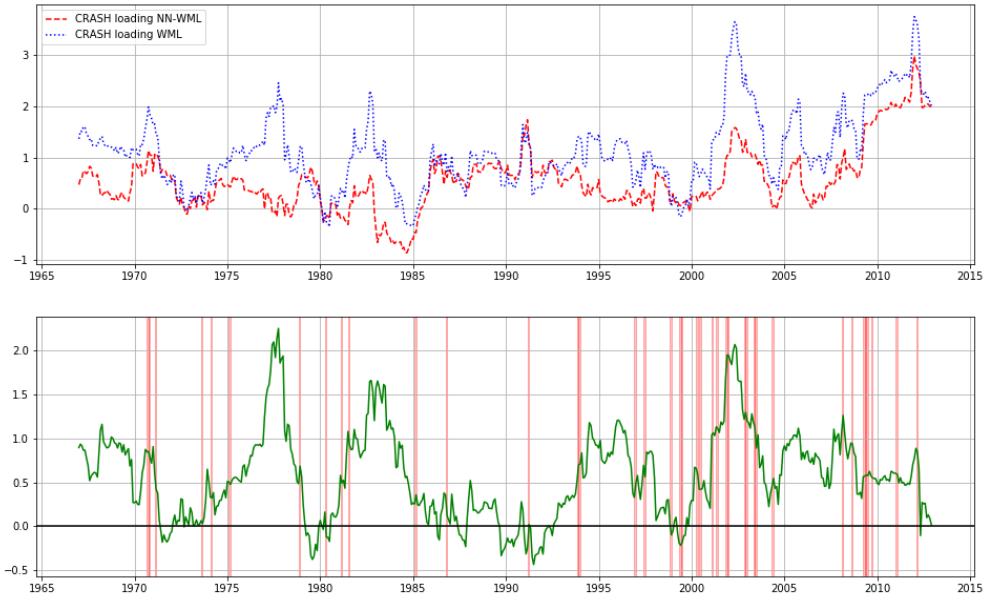
	factor	SL	SW	BL	BW
Mean return	1.66	-0.44	2.20	0.46	1.15
t-statistic	9.38	-1.38	8.85	1.95	6.00
σ	4.30	7.81	6.02	5.76	4.64
Sharpe ratio (ann.)	1.34	-0.20	1.26	0.28	0.86
Median	1.97	-0.51	2.59	0.40	1.43
Min	-27.33	-26.91	-32.45	-25.78	-23.30
Max	21.89	45.52	26.62	22.24	19.28
2%-percentile	-10.08	-17.55	-10.60	-12.96	-9.23
5%-percentile	-4.44	-11.36	-7.85	-8.67	-6.07

Results in table 10 are in line with the findings in the previous section, suggesting that the performance of the neural network momentum is partly driven by short positions in small-cap stocks. However, factor-like formulation still generates Sharpe ratio of 1.34 beating both the benchmark momentum strategy (Sharpe ratio 0.73) and also the risk managed momentum strategies by Barroso and Santa-Clara (2015) and Daniel and Moskowitz (2016) (1.041 and 1.194).

5.6 Time varying tail-risk

To analyze further the risk characteristics of the NN-WML and WML portfolios, I study the time-varying loading of the momentum portfolios on the CRASH factor by Chabi-Yo et al. (2018). Figure 5 shows the neural network and benchmark momentum portfolios' rolling loadings on the CRASH factor in the first plot and spread between the loadings

Figure 5: The first figure shows rolling loading of NN-WML and WML portfolios on the CRASH factor by Chabi-Yo et al. (2018). Loading is estimated by regressing momentum portfolios' returns on Fama-French three factor model with CRASH factor. I use 36 month rolling window. Second subfigure plots the spread between the CRASH loadings of WML and NN-WML. Red bars indicate months with WML return below -10%



in the second plot. Precisely, in the time period from January 1964 to December 2012, I use 36 month rolling window to regress the NN-WML and WML returns on the excess market return, size, value and CRASH factors, i.e., the Fama-French three factor model with the CRASH factor. In the second plot, I have highlighted the months where the WML return was below -10% by shaded bars. Correlation between the rolling loadings is 74% and mean of the spread is 0.56 with t-statistic of 13.66 indicating that the hypothesis of a zero spread can be rejected at a very high confidence level.

For most of the time during the estimation window, the CRASH loading of the WML stays higher than the loading of the NN-WML, meaning that the spread between the loadings also remains positive. Both time series show occasional jumps and periods of higher exposure to the market wide crash risk. In particular, the CRASH loading of the WML is at a very high levels after the dot-com bubble crash in early 2000s, when the market started to rebound. During that time, the loading on the NN-WML also spiked but not as much the loading of the WML. A similar pattern is distinguishable during the subprime crisis in 2008 and 2009. The loading of the NN-WML also increases greatly during this period and is much closer to the WML loading compared to the post dot-

com bubble. The spread between the loadings also widens heavily between 1976-1979 and 1982-1984. However, during these periods, the widening was mostly caused by the increase on the WML CRASH loading.

These findings support the results earlier and show that on average, the WML exhibits stronger tail risk than the NN-WML. Despite of this, the CRASH loading of the NN-WML is also positive and shows occasional spikes in the times when momentum strategy has been documented to suffer the worst losses, indicating that neural networks help only to reduce tail risk in the momentum strategy, not remove it.

6 Feature importances

One interesting question is which features drive the predictions the neural network makes. Unfortunately, interpretation of the feature importances with the neural networks is hard compared, for instance, to linear regression models which provide a simple and intuitive interpretation in the form of regression coefficients. Probably the most robust way to measure to which degree each of the features drive the predictions would be to drop each of the features one at time, train the model and then compare quality of the predictions to the situation where all the features are used. However, when the datasets grow larger, such methodology becomes very fast computationally infeasible. Hence, some kind of approximation, or a proxy needs to be constructed.

To measure feature importances, I follow the approach by Messmer (2017). I separately set each of the feature vectors to zero and then measure the change in predictions and compare it to the situation in which all the features are used. The particular measure I use is the mean squared deviation (MSD) which is calculated as follows

$$MSD_{t,k} = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{y}_{i,t} - \hat{y}_{i,t,x_{i,t,k}=0} \right)^2 \tag{15}$$

Here, $\hat{y}_{i,t}$ is the prediction made for stock i at time t using all features. In turn, $\hat{y}_{i,t,x_{i,t,k}=0}$ is the prediction after setting feature k to zero. Obviously, such specification to measure feature importance is not a perfect proxy of the feature importance.

Next, I pool estimates to yearly level and scale each measure into the interval from 0 to 1 such that the sum of feature contributions is 1. Moreover, I aggregate contributions of the momentum signals into three categories: short-, mid- and long-term momentum for illustrative purposes. Short-term momentum contribution is sum of 2, 3 and 4 momentum contributions; mid term sum of 5, 6, 7 and 8; and long term sum of 9, 10, 11 and 12 month momentum signals. Figure 6 shows these pooled estimates over the strategy estimation

period from 1964 to 2012. The vertical lines every fifth year indicate years when the model is re-trained.

Not surprisingly, long term-momentum signals are the most important features trough out the sample, followed by mid- and short-term momentum signals. Next comes one and six month rolling volatility estimates. This is also the expected result as previous research shows that volatility scaling improves momentum strategy performance substantially. Shorter term volatilities seem to be more informative for the neural network than the one year rolling volatility through the sample. Given the mean-reverting nature of the volatility, it is likely that the longer volatility estimates are more stable and provide less infromation than the short and mid term estimates. Tail dependence measures, LTD and UTD, are the least important features with the one year volatility. Hence, it seems that the MSD measures provide at least information about the cross-sectional importances between the features.

However, time variation in the MSD-based feature importances seems to be almost non-existent. Most of the variation is driven by the re-training of the model which is seen from the figure 6 where the absolute level of the MSD estimates jumps every fifth year. There neither is distinguishable spikes in the risk indicator importances during years when momentum crashes occur.

These results highlight the limitations of the use of neural networks: interpreting causal, or even statistical relations with outputs and inputs is a rather hard exercise. I expect that training the model more frequently would allow to distinguish more time variation in the feature importances, but given that the computational burden of fitting neural networks, it was not feasible in my case.

Figure 6: Annual feature importance estimates based on mean squared deviation (MSD) of the predictions when the given feature vector k is set to zero. Feature importances are normalized each year such that the sum of them equals 1. 2, 3 and 4; 5, 6, 7 and 8; 9, 10, 11 and 12 month month momentum signals are pooled under short, mid and long-momentum respectively. Vertical lines indicate years when the neural network is retrained.

	LTD	UTD	Long mom	Mid mom	Short mom	vola_30	vola_126	vola_252
1964	0.043	0.041	0.250	0.188	0.201	0.095	0.076	0.041
1965	0.043	0.041	0.250	0.188	0.199	0.097	0.077	0.041
1966	0.042	0.040	0.252	0.190	0.197	0.098	0.076	0.040
1967	0.042	0.040	0.257	0.193	0.197	0.094	0.072	0.040
1968	0.043	0.041	0.257	0.194	0.197	0.092	0.069	0.042
1969	0.037	0.037	0.262	0.197	0.203	0.093	0.068	0.037
1970	0.037	0.037	0.259	0.196	0.206	0.092	0.068	0.037
1971	0.037	0.037	0.260	0.195	0.202	0.096	0.071	0.037
1972	0.037	0.037	0.258	0.194	0.202	0.097	0.071	0.037
1973	0.037	0.037	0.257	0.195	0.207	0.093	0.069	0.037
1974	0.040	0.041	0.260	0.196	0.199	0.088	0.070	0.040
1975	0.040	0.040	0.261	0.196	0.195	0.090	0.072	0.040
1976	0.039	0.040	0.260	0.194	0.194	0.093	0.074	0.040
1977	0.038	0.039	0.261	0.195	0.196	0.092	0.074	0.039
1978	0.038	0.039	0.262	0.195	0.194	0.095	0.074	0.038
1979	0.046	0.049	0.264	0.197	0.187	0.081	0.068	0.045
1980	0.046	0.049	0.263	0.197	0.189	0.081	0.067	0.045
1981	0.046	0.049	0.263	0.197	0.188	0.080	0.067	0.045
1982	0.043	0.045	0.265	0.199	0.195	0.078	0.065	0.042
1983	0.045	0.047	0.265	0.200	0.191	0.078	0.066	0.044
1984	0.041	0.042	0.260	0.192	0.197	0.090	0.073	0.042
1985	0.040	0.041	0.261	0.193	0.198	0.088	0.072	0.041
1986	0.040	0.041	0.260	0.192	0.196	0.091	0.074	0.041
1987	0.040	0.041	0.261	0.192	0.194	0.092	0.074	0.041
1988	0.041	0.042	0.261	0.193	0.195	0.089	0.072	0.042
1989	0.042	0.048	0.254	0.201	0.200	0.079	0.070	0.041
1990	0.042	0.048	0.253	0.202	0.201	0.079	0.069	0.041
1991	0.043	0.049	0.252	0.199	0.199	0.080	0.071	0.042
1992	0.045	0.053	0.251	0.191	0.191	0.087	0.077	0.043
1993	0.045	0.053	0.246	0.191	0.191	0.090	0.078	0.043
1994	0.053	0.059	0.239	0.195	0.198	0.074	0.064	0.051
1995	0.052	0.057	0.239	0.197	0.201	0.074	0.063	0.049
1996	0.053	0.059	0.241	0.193	0.197	0.077	0.065	0.050
1997	0.051	0.057	0.241	0.200	0.204	0.070	0.060	0.049
1998	0.051	0.056	0.242	0.200	0.203	0.070	0.060	0.049
1999	0.043	0.046	0.257	0.202	0.202	0.080	0.060	0.043
2000	0.046	0.050	0.259	0.189	0.187	0.094	0.066	0.046
2001	0.041	0.043	0.262	0.213	0.202	0.072	0.055	0.041
2002	0.041	0.044	0.261	0.209	0.202	0.074	0.056	0.042
2003	0.042	0.045	0.254	0.206	0.207	0.075	0.058	0.043
2004	0.038	0.037	0.267	0.206	0.201	0.081	0.063	0.040
2005	0.037	0.037	0.259	0.216	0.215	0.069	0.056	0.039
2006	0.038	0.038	0.263	0.213	0.209	0.073	0.058	0.040
2007	0.038	0.038	0.259	0.216	0.214	0.069	0.056	0.039
2008	0.037	0.037	0.260	0.224	0.211	0.066	0.054	0.039
2009	0.037	0.037	0.261	0.214	0.216	0.070	0.054	0.038
2010	0.036	0.036	0.267	0.208	0.210	0.078	0.057	0.037
2011	0.037	0.036	0.263	0.213	0.214	0.073	0.055	0.037
2012	0.037	0.037	0.259	0.214	0.217	0.070	0.054	0.038

7 Conclusion

Machine learning and, in particular, neural networks are powerful tools for predictive analysis and are used in wide range of applications. Their use in the field of empirical finance has also received some attention in the financial literature lately. In this paper, I study how neural networks can be used to enhance cross-sectional momentum investment strategy. Using momentum signals and stock level risk indicators, I train neural network to predict relative future performance of US stocks between 1964 and 2012. Based on the predictions, I construct a long-short neural network momentum portfolio (NN-WML).

I show that the performance of neural network enhanced momentum strategy ouperformns the traditional momentum strategy (WML) by a wide margin. NN-WML and WML portfolios generate average monthly returns of 3.18% (t-stat 11.81) and 1.79% (t-stat 5.12) and annualized Sharpe ratios of 1.69 and 0.73 respectively. Returns on the NN-WML remain robust after controlling for conventional risk factors. Moreover, after controlling for benchmark momentum, the NN-WML still generates a significant alpha of 2.3%. Conversely, the alpha of the WML disappears after controlling for the NN-WML. Results are robust for limiting the stock universe to contain only larger firms and for factor-like formulation of the strategy, such that the size effect is filtered out. I also show that use of neural networks helps to mitigate tail risk that momentum strategy has been documented to have.

I also study which of the features are driving the predictions the neural network makes. Not surprisingly, I find that the long term momentum signals are the most important features. Also consistent with the existing literature, which suggests that volatility scaling improves momentum strategy performance significantly, short and mid term volatility estimates are important drivers of the predictions.

Machine learning methods will with a high probability play large role in the future of the asset management industry. Although machine learning has been used in previous research there still lies big questions that need answers. As my paper, also related research has been conducted with data before 2012, it will be interesting to see how the machine learning based strategies perform in the time, when they are actually being used more and more. Moreover, finding a way to infer feature importances in a similar way as regression loadings would provide wider range of applications in which machine learning could be combined with empirical finance research.

A Appendices

A.1 Adaptive moment estimation algorithm

This appendix provides details for the ADAM algorithm (Kingma & Ba, 2014) and gives description of the steps needed to implement the algorithm. General parameters for ADAM are the learning rate α and exponential decay rates $\beta_1, \beta_2 \in [0, 1]$. Learning rate α is treated as an optimizable hyperparameter, as described in the main text. Algorithm 1 gives details on the ADAM implementation:

Algorithm 1 Adaptive moment estimation (ADAM) algorithm.

1: **Initialize:**
2: $k = 0$
3: $\theta = \theta_0$
4: $m_0 = 0$
5: $s_0 = 0$
6: **while** Not terminated by early stopping **do**
7: Draw random batch $\{X_{batch}, y_{batch}\} \in \{X_{train}, y_{train}\}$
8: $k = k + 1$
9: $g_k = \nabla_\theta \hat{L}(X_{batch}, y_{batch}, \theta; \phi)$
10: $m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$
11: $s_k = \beta_1 s_{k-1} + (1 - \beta_1) g_k^2$
12: $\hat{m}_k = \dfrac{m_k}{1 - \beta_1^k}$
13: $\hat{s}_k = \dfrac{s_k}{1 - \beta_1^k}$
14: $\theta_k = \theta_{k-1} - \alpha \dfrac{\hat{m}_k}{\sqrt{\hat{v}_k} + \varepsilon}$
15: **end while**
16: **Return**

In line 11 of algorithm 1 the second power is applied to the gradient element wise. Similarly, in line 14, division is made elementwise. Operations in the lines 12 and 13 are called bias correction. I motivate the bias correction for the exponential moving average of the gradient, m_k. The argumentation for s_k is analogous. Note that m_k can be written recursively as

$$
\begin{aligned}
m_0 &= 0 \\
m_1 &= \beta_1 m_0 + (1 - \beta_1) g_1 = (1 - \beta_1) g_1 \\
m_2 &= \beta_1 (1 - \beta_1) g_1 + (1 - \beta_1) g_2 \\
m_3 &= \beta_1^2 (1 - \beta_1) g_1 + \beta_1 (1 - \beta_1) g_2 + (1 - \beta_1) g_3 \\
&\vdots
\end{aligned}
$$

$$m_k = (1 - \beta_1) \sum_{i=0}^{k} g_i \beta_1^{k-i}$$

Taking expectation of m_k gives

$$E[m_k] = E\left[(1 - \beta_1) \sum_{i=0}^{k} g_i \beta_1^{k-i}\right] \tag{16}$$

Next, make approximation $E[g_i] \approx E[g_k]$ and substitute to (16):

$$E[m_k] = E[g_k](1 - \beta_1) \sum_{i=0}^{k} \beta_1^{k-i} + \xi \tag{17}$$

$$E[m_k] = E[g_k](1 - \beta_1^k) + \xi \tag{18}$$

Where $\xi = 0$ if g_k is stationary. Where the last equality follows from the properties of finite geometric series. Dividing by $(1 - \beta_1^k)$ gives bias corrected estimate $\hat{m}_k$ of g_k.

A.2 Early stopping algorithm

For early stopping algorithm, I follow the specification by Goodfellow et al. (2016). Algorithm proceeds as follows.

Algorithm 2 Early stopping algorithm. Let n be the number of steps between evaluations, p the patience parameter, θ_0 initial parameter guess and θ^* set of the best parameters

Initialize:
$\theta = \theta_0$
$i = 0$
$j = 0$
$v^* = \infty$ ▷ Best validation error
$\theta^* = \theta$
$i^* = \mathrm{i}$
while j < p **do**
    Update θ by running ADAM algorithm for n steps
    $i = i + n$
    $v = \hat{L}(X_{val}, y_{val}, \theta; \phi)$
    if $v < v^*$ **then**
        $j = 0$
        $\theta^* = \theta$
        $i^* = i$
        $v^* = v$
    else
        $j = j + 1$
    end if
end while
Return θ^*

A.3 Equal weighted momentum portfolios

Table 11 reports same statistics as table 4 but the portfolios are equal weighted. Somewhat surprisingly, using equal weighting, benchmark momentum strategy's average monthly actually decreases approximately by 1%-point from 1.79% to 0.78%. Conversely, neural network momentum strategy average return is almost doubles from 3.18% to 5.93%.

Table 11: Summary statistics for equal weighted benchmark momentum and neural network based cross-sectional momentum strategies. The time period is January 1964 - December 2012. Numbers are reported in percentages and in monthly basis. First ten columns of panels A and B represent decile portfolios where column 1 (10) consists of stocks whose momentum signal is lowest (highest). Last columns, WML and NN-WML, are a zero-investment long-short portfolio buying portfolio 10 and selling portfolio 1.

Panel A: Summary statistics for equal weighted benchmark momentum strategy returns

	1	2	3	4	5	6	7	8	9	10	WML
Mean return	1.05	0.84	0.93	1.08	1.11	1.21	1.34	1.49	1.65	1.84	0.78
σ	10.36	7.49	6.38	5.68	5.17	5.04	5.03	5.19	5.79	7.29	7.81
Sharpe ratio	0.10	0.11	0.15	0.19	0.21	0.24	0.27	0.29	0.28	0.25	0.10
Median	0.44	0.73	1.21	1.29	1.49	1.68	1.84	1.97	1.94	2.36	1.41
Min	-29.69	-27.60	-26.67	-24.01	-25.03	-26.62	-27.08	-28.16	-29.43	-32.60	-80.79
Max	85.66	52.17	34.00	34.52	29.62	26.10	24.35	23.00	24.28	37.16	27.20
2%-percentile	-17.55	-13.24	-12.19	-10.52	-10.29	-9.86	-9.47	-10.14	-11.55	-14.76	-19.09
5%-percentile	-13.47	-10.63	-9.22	-7.67	-7.07	-6.73	-7.19	-7.04	-8.29	-10.34	-11.33

Panel B: Summary statistics for neural equal weighted network momentum strategy returns.

	1	2	3	4	5	6	7	8	9	10	WML
Mean return	-1.80	-0.08	0.79	1.17	1.34	1.46	1.59	1.81	2.13	4.13	5.93
σ	7.46	7.61	7.60	6.60	5.88	5.57	5.30	5.27	5.48	7.47	6.68
Sharpe ratio	-0.24	-0.01	0.10	0.18	0.23	0.26	0.30	0.34	0.39	0.55	0.89
Median	-2.01	-0.37	0.52	1.11	1.48	1.86	2.03	2.11	2.32	3.52	6.28
Min	-26.76	-27.10	-25.52	-26.58	-24.57	-28.35	-31.05	-29.99	-30.09	-31.42	-47.67
Max	48.09	54.71	56.36	37.47	27.93	27.85	27.17	26.46	27.20	35.98	23.85
2%-percentile	-18.38	-16.38	-13.84	-12.87	-11.58	-11.06	-10.58	-9.36	-10.03	-10.84	-10.34
5%-percentile	-13.11	-11.32	-9.63	-8.69	-8.01	-7.33	-7.18	-6.79	-6.98	-6.25	-4.19

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `http://tensorflow.org/` (Software available from tensorflow.org)

Asness, C. S. (1997). The interaction of value and momentum strategies. *Financial Analysts Journal*, *53*(2), 29–36.

Asness, C. S., Moskowitz, T. J., & Pedersen, L. H. (2013). Value and momentum everywhere. *The Journal of Finance*, *68*(3), 929–985.

Barroso, P., & Santa-Clara, P. (2015). Momentum has its moments. *Journal of Financial Economics*, *116*(1), 111–120.

Chabi-Yo, F., Ruenzi, S., & Weigert, F. (2018). Crash sensitivity and the cross section of expected stock returns. *Journal of Financial and Quantitative Analysis*, *53*(3), 1059–1100.

Cybenko, G. (1989). Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*, 183–192.

Daniel, K., Hirshleifer, D., & Subrahmanyam, A. (1998). Investor psychology and security market under-and overreactions. *the Journal of Finance*, *53*(6), 1839–1885.

Daniel, K., & Moskowitz, T. J. (2016). Momentum crashes. *Journal of Financial Economics*, *122*(2), 221–247.

Duan, Y., Hu, G., & McLean, R. D. (2010). Costly arbitrage and idiosyncratic risk: Evidence from short sellers. *Journal of Financial Intermediation*, *19*(4), 564–579.

Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of financial economics*, *33*(1), 3–56.

Feng, G., Giglio, S., & Xiu, D. (2017). *Taming the factor zoo.* (Chicago Booth Research Paper No. 17-04)

George, T. J., & Hwang, C.-Y. (2004). The 52-week high and momentum investing. *The Journal of Finance*, *59*(5), 2145–2176.

Goetzmann, W. N., & Huang, S. (2018). Momentum in imperial russia. *Journal of Financial Economics*, *130*(3), 579–591.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press. (`http://www.deeplearningbook.org`)

Grinblatt, M., & Han, B. (2005). Prospect theory, mental accounting, and momentum. *Journal of financial economics*, *78*(2), 311–339.

Gu, S., Kelly, B. T., & Xiu, D. (2018). *Empirical asset pricing via machine learning.* (Chicago Booth Research Paper No. 18-04)

Gu, S., Kelly, B. T., & Xiu, D. (2019). Autoencoder asset pricing models. *Available at SSRN*.

Hill, B. M. (1975). A simple general approach to inference about the tail of a distribution. *The annals of statistics*, 1163–1174.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, *4*(2), 251–257.

Jegadeesh, N. (1990). Evidence of predictable behavior of security returns. *The Journal of finance*, *45*(3), 881–898.

Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of finance*, *48*(1), 65–91.

Johnson, T. C. (2002). Rational momentum effects. *The Journal of Finance*, *57*(2), 585–608.

Kelly, B., & Jiang, H. (2014). Tail risk and asset prices. *The Review of Financial Studies*, *27*(10), 2841–2871.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lehmann, B. N. (1990). Fads, martingales, and market efficiency. *The Quarterly Journal*

of Economics, *105*(1), 1–28.

Messmer, M. (2017). *Deep learning and the cross-section of expected returns.* (Doctoral dissertation, University of St. Gallen)

Messmer, M., & Audrino, F. (2017). The (adaptive) lasso in the zoo-firm characteristic selection in the cross-section of expected returns.

Moskowitz, T. J., Ooi, Y. H., & Pedersen, L. H. (2012). Time series momentum. *Journal of financial economics*, *104*(2), 228–250.

Newey, W. K., & West, K. D. (1986). *A simple, positive semi-definite, heteroskedasticity and autocorrelationconsistent covariance matrix.* National Bureau of Economic Research Cambridge, Mass., USA.

Ruenzi, S., & Weigert, F. (2018). Momentum and crash sensitivity. *Economics Letters*, *165*, 77–81.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Takeuchi, L., & Lee, Y.-Y. A. (2013). *Applying deep learning to enhance momentum trading strategies in stocks* (Tech. Rep.).